Display Lists, Viewing

Week 3, Fri Jan 21

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2005
Reading

- RB Chapter Display Lists
  - (it’s short)
- Viewing: FCG Section 6.2.1
  - Arbitrary View Positions
Project 1 Clarifications, Hints

- finish all required parts before
  - going for extra credit
  - playing with lighting or viewing
- ok to use glRotate, glTranslate, glScale
- ok to use glutSolidCube, or build your own
  - where to put origin? your choice
    - center of object, range - .5 to +.5
    - corner of object, range 0 to 1
Project 1 Clarifications, Hints

- visual debugging
  - color cube faces differently
  - colored lines sticking out of glutSolidCube faces
- thinking about transformations
  - move physical objects around
  - play with demos
    - Brown scenegraph applets
Project 1 Clarifications, Hints

- transitions
  - safe to linearly interpolate parameters for `glRotate/glTranslate/glScale`
  - do not interpolate individual elements of 4x4 matrix!
Review: Transformation Hierarchies

- transforms apply to graph nodes beneath them
- design structure so that object doesn’t fall apart
Review: Matrix Stacks

- OpenGL matrix calls postmultiply matrix M onto current matrix P, overwrite it to be PM
- or can save intermediate states with stack

```
glPushMatrix

A
A

B
B

C
C

A
A

glPopMatrix
```

```
D = C scale(2,2,2) trans(1,0,0)
```

```
glPushMatrix

C
C

B
B

A
A

glPopMatrix
```

```
C
C

D
D

C
C

A
A

glPopMatrix
```

```
C
C

D
D

C
C

A
A

glPopMatrix
```

```
D = C scale(2,2,2) trans(1,0,0)
```

```
DrawSquare()

glPushMatrix()

glScale3f(2,2,2)

glTranslate3f(1,0,0)

DrawSquare()

glPopMatrix()
```
Matrix Stacks

- advantages
  - no need to compute inverse matrices all the time
  - modularize changes to pipeline state
  - avoids incremental changes to coordinate systems
    - accumulation of numerical errors

- practical issues
  - in graphics hardware, depth of matrix stacks is limited
    - (typically 16 for model/view and about 4 for projective matrix)
Hierarchical Modelling

- advantages
  - define object once, instantiate multiple copies
  - transformation parameters often good control knobs
  - maintain structural constraints if well-designed

- limitations
  - expressivity: not always the best controls
  - can’t do closed kinematic chains
    - keep hand on hip
  - can’t do other constraints
    - collision detection
      - self-intersection
      - walk through walls
Single Parameter: Simple

- parameters as functions of other parameters
  - clock: control all hands with seconds $s$

  $m = s/60$, $h = m/60$,
  $\theta_s = (2 \pi s) / 60$,
  $\theta_m = (2 \pi m) / 60$,
  $\theta_h = (2 \pi h) / 60$
Single Parameter: Complex

- mechanisms not easily expressible with affine transforms

http://www.flying-pig.co.uk
http://www.flying-pig.co.uk/mechanisms/pages/irregular.html
Display Lists
Display Lists

- reuse block of OpenGL code
  - more efficient than immediate mode
    - avoid function calls for every vertex/attribute, driver optimization, graphics board cache (bandwidth!)
  - good for multiple instances of same object
    - but cannot change contents, not parametrizable
  - good for static objects redrawn often
    - display lists persist across multiple frames
    - interactive graphics: objects redrawn every frame from new viewpoint from moving camera
- can be nested hierarchically

- snowman example
  http://www.lighthouse3d.com/opengl/displaylists
void drawSnowMan() {

    glColor3f(1.0f, 1.0f, 1.0f);

    // Draw Body
    glTranslatef(0.0f, 0.75f, 0.0f);
    glutSolidSphere(0.75f, 20, 20);

    // Draw Head
    glTranslatef(0.0f, 1.0f, 0.0f);
    glutSolidSphere(0.25f, 20, 20);

    // Draw Nose
    glColor3f(1.0f, 0.5f, 0.5f);
    glRotatef(0.0f, 1.0f, 0.0f, 0.0f);
    glutSolidCone(0.08f, 0.5f, 10, 2);

    // Draw Eyes
    glPushMatrix();
    glColor3f(0.0f, 0.0f, 0.0f);
    glTranslatef(0.05f, 0.10f, 0.18f);
    glutSolidSphere(0.05f, 10, 10);
    glTranslatef(-0.1f, 0.0f, 0.0f);
    glutSolidSphere(0.05f, 10, 10);
    glPopMatrix();
}

// Draw Snowman

drawSnowMan()
Snowmen: No Lists

// Draw 36 Snowmen
for(int i = -3; i < 3; i++)
    for(int j=-3; j < 3; j++) {
        glPushMatrix();
        glTranslatef(i*10.0,0,j * 10.0);
        // Call the function to draw a snowman
drawSnowMan();
        glPopMatrix();
    }

36K polygons, 55 FPS
Making Display Lists

```cpp
GLuint createDL() {
    GLuint snowManDL;
    // Create the id for the list
    snowManDL = glGenLists(1);
    // start list
    glNewList(snowManDL, GL_COMPILE);
    // call the function that contains the rendering commands
    drawSnowMan();
    // endList
    glEndList();
    return(snowManDL); }
```
Snowmen: Display Lists

// Draw 36 Snowmen

for(int i = -3; i < 3; i++)
    for(int j=-3; j < 3; j++) {
        glPushMatrix();
        glTranslatef(i*10.0,0,j * 10.0);
        // Call the function to draw a snowman
        glCallList(Dlid);
        glPopMatrix();
    }

153 FPS
GLuint createDL() {
    GLuint snowManDL;
    snowManDL = glGenLists(1);
    glNewList(snowManDL, GL_COMPILE);
    for(int i = -3; i < 3; i++)
        for(int j = -3; j < 3; j++) {
            glPushMatrix();
            glTranslatef(i*10.0,0,j * 10.0);
            drawSnowMan();
            glPopMatrix();
        }
    glEndList();
    return(snowManDL);
}

108 FPS
Snowmen: Hierarchical Lists

GLuint createDL() {
    GLuint snowManDL, loopDL;
    snowManDL = glGenLists(1);
    loopDL = glGenLists(1);
    glNewList(snowManDL, GL_COMPILE);
    drawSnowMan();
    glEndList();
    glNewList(loopDL, GL_COMPILE);
    for (int i = -3; i < 3; i++)
        for (int j = -3; j < 3; j++) {
            glPushMatrix();
            glTranslatef(i * 10.0, 0, j * 10.0);
            glCallList(snowManDL);
            glPopMatrix();
        }
    glEndList();
    return (loopDL); }

153 FPS
Display Lists

- example: 36 snowmen
  - small display list with 36x reuse
    - 3x faster
  - big display list with 1x reuse
    - 2x faster
- nested display lists, 1x * 36x reuse:
  - 3x faster, high-level block available
  - exploit hierarchical structure
Viewing and Projection
Using Transformations

- three ways
  - modelling transforms
    - place objects within scene (shared world)
  - viewing transforms
    - place camera
  - projection transforms
    - change type of camera
Viewing and Projection

- need to get from 3D world to 2D image
- projection: geometric abstraction
  - what eyes or cameras do
- two pieces
  - viewing transform:
    - where is the camera, what is it pointing at?
  - perspective transform: 3D to 2D
    - flatten to image
Rendering Pipeline

Geometry Database → Model/View Transform. → Lighting → Perspective Transform. → Clipping

Scan Conversion → Texturing → Depth Test → Blending → Frame-buffer
Rendering Pipeline

Geometry Database → Model/View Transform. → Lighting → Perspective Transform. → Clipping

Scan Conversion → Texturing → Depth Test → Blending → Frame-buffer
Rendering Pipeline

- Scene graph
- Object geometry
- Modelling
- Transforms
- Viewing
- Transform
- Projection
- Transform
Rendering Pipeline

- result
  - all vertices of scene in shared 3D world coordinate system
Rendering Pipeline

- **result**
  - scene vertices in 3D view (camera) coordinate system

---

Scene graph
Object geometry

**Modelling**
Transforms

**Viewing**
Transform

**Projection**
Transform
Rendering Pipeline

- result
  - 2D screen coordinates of clipped vertices
Coordinate Systems

- result of a transformation
- names
  - convenience
    - kangaroo: neck, head, tail
  - standard conventions in graphics pipeline
    - object/modelling
    - world
    - camera/viewing/eye
    - screen/window
    - raster/device
Projective Rendering Pipeline

OCS - object/model coordinate system
WCS - world coordinate system
VCS - viewing/camera/eye coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device/display/screen coordinate system

modeling transformation
viewing transformation
projection transformation

perspective divide
viewport transformation

clipping
CCS
normalized
device
NDCS
device
DCS
Basic Viewing

- starting spot - OpenGL
  - camera at world origin
    - probably inside an object
  - y axis is up
  - looking down negative z axis
    - why? RHS with x horizontal, y vertical, z out of screen
- translate backward so scene is visible
  - move distance \( d = \text{focal length} \)
- can use rotate/translate/scale to move camera
  - demo: Nate Robins tutorial *transformations*
Viewing in Project 1

- where is camera in template code?
  - 5 units back, looking down -z axis
Convenient Camera Motion

- rotate/translate/scale not intuitive
- arbitrary viewing position
  - eye point, gaze/lookat direction, up vector
Convenient Camera Motion

- rotate/translate/scale not intuitive
- arbitrary viewing position
  - eye point, gaze/lookat direction, up vector
From World to View Coordinates

- translate **eye** to origin
- rotate **view** vector (**lookat** – **eye**) to **w** axis
- rotate around **w** to bring **up** into **vw**-plane
OpenGL Viewing Transformation

```c
gluLookAt(ex, ey, ez, lx, ly, lz, ux, uy, uz)
```

- postmultiplies current matrix, so to be safe:

```c
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(ex, ey, ez, lx, ly, lz, ux, uy, uz)
// now ok to do model transformations
```

- demo: Nate Robins tutorial  projection
Deriving World-to-View Transformation

- translate \textbf{eye} to origin

\[
T = \begin{bmatrix}
1 & 0 & 0 & -e_x \\
0 & 1 & 0 & -e_y \\
0 & 0 & 1 & -e_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Deriving World-to-View Transformation

- rotate view vector (lookat - eye) to w axis
- \( w \) is just opposite of view/gaze vector \( g \)

\[
w = -\hat{g} = -\frac{g}{\|g\|}
\]
Deriving World-to-View Transformation

- rotate around \( w \) to bring \textit{up} into \( vw \)-plane
  - \( u \) should be perpendicular to \( vw \)-plane, thus perpendicular to \( w \) and \textit{up} vector \( t \)
  - \( v \) should be perpendicular to \( u \) and \( w \)

\[
\begin{align*}
u &= \frac{t \times w}{\|t \times w\|} \\
v &= w \times u
\end{align*}
\]
Deriving World-to-View Transformation

- rotate from WCS $\text{xyz}$ into $\text{uvw}$ coordinate system with matrix that has rows $u, v, w$

$$u = \frac{t \times w}{\|t \times w\|} \quad v = w \times u \quad w = -\hat{g} = -\frac{g}{\|g\|}$$

$$R = \begin{bmatrix}
  u_x & u_y & u_z & 0 \\
  v_x & v_y & v_z & 0 \\
  w_x & w_y & w_z & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}$$

- reminder: rotate from $\text{uvw}$ to $\text{xyz}$ coord sys with matrix $\textbf{M}$ that has columns $u,v,w$
  - rotate from $\text{xyz}$ coord sys to $\text{uvw}$ coord sys with matrix $\textbf{M}^T$ that has rows $u,v,w$
Deriving World-to-View Transformation

\[ M = RT \]

\[ \begin{bmatrix}
  u_x & u_y & u_z & 0 \\
  v_x & v_y & v_z & 0 \\
  w_x & w_y & w_z & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix} =
\begin{bmatrix}
  1 & 0 & 0 & -e_x \\
  0 & 1 & 0 & -e_y \\
  0 & 0 & 1 & -e_z \\
  0 & 0 & 0 & 1
\end{bmatrix} \]

\[ M_{\text{world} \rightarrow \text{view}} =
\begin{bmatrix}
  u_x & u_y & u_z & 0 \\
  v_x & v_y & v_z & 0 \\
  w_x & w_y & w_z & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  1 & 0 & 0 & -e_x \\
  0 & 1 & 0 & -e_y \\
  0 & 0 & 1 & -e_z \\
  0 & 0 & 0 & 1
\end{bmatrix} =
\begin{bmatrix}
  u_x & u_y & u_z & -u \cdot e \\
  v_x & v_y & v_z & -v \cdot e \\
  w_x & w_y & w_z & -w \cdot e \\
  0 & 0 & 0 & 1
\end{bmatrix} \]
Moving the Camera or the World?

- two equivalent operations
  - move camera one way vs. move world other way
- example
  - initial OpenGL camera: at origin, looking along -z axis
  - create a unit square parallel to camera at z = -10
  - translate in z by 3 possible in two ways
    - camera moves to z = -3
      - Note OpenGL models viewing in left-hand coordinates
    - camera stays put, but square moves to -7
  - resulting image same either way
    - possible difference: are lights specified in world or view coordinates?